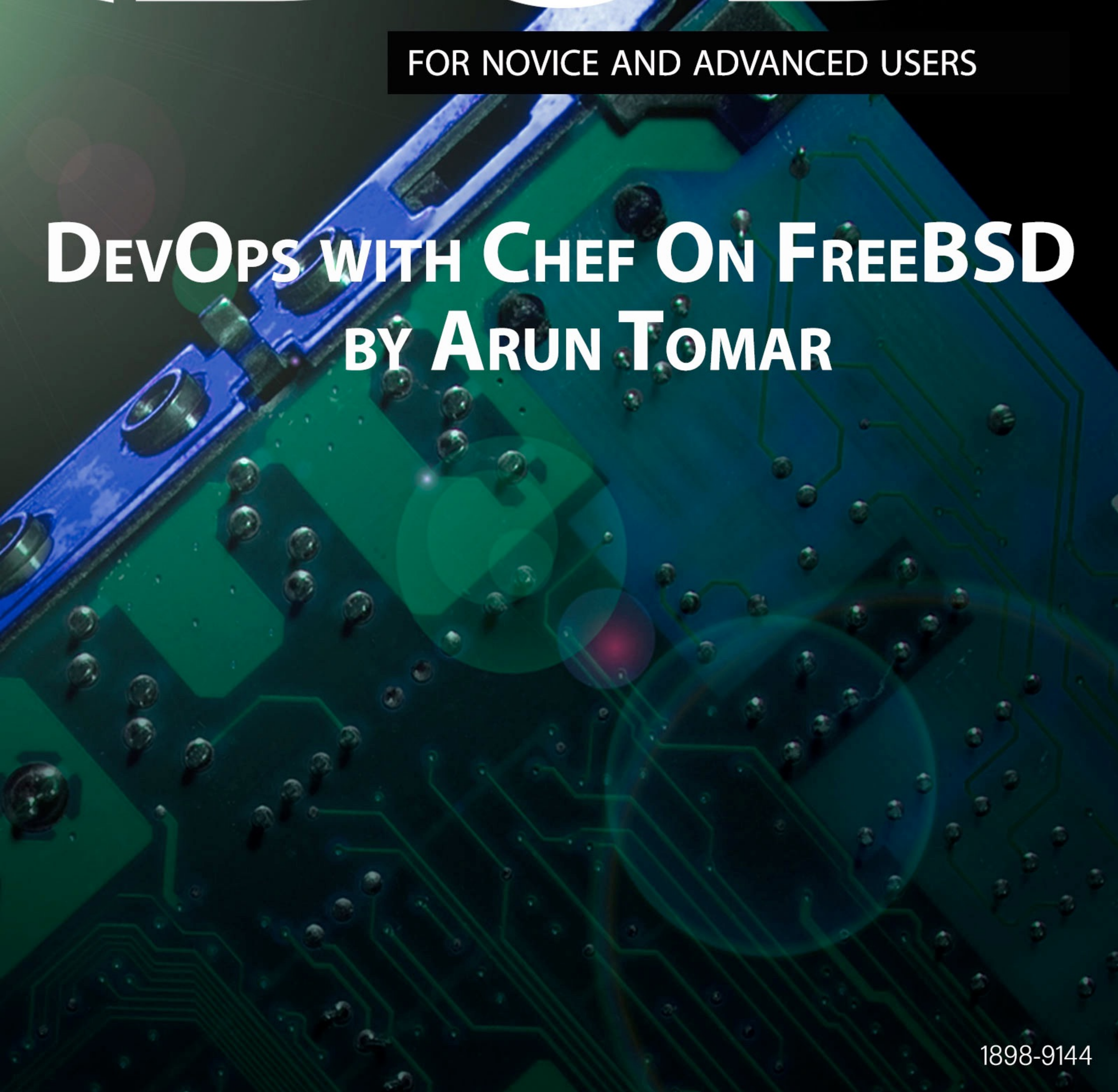
BSD MAGAZINE

FOR NOVICE AND ADVANCED USERS

DevOps with Chef On FreeBSD by Arun Tomar



1898-9144

DevOps with Chef on FreeBSD

Dear Participants,

Welcome to the course DevOps with Chef on FreeBSD. In this course, we'll cover lot of theory and practical aspects. We'll focus on the core topics as mentioned in the agenda, and we assume that you already know the prerequisites. If not, then you might have to learn additional topics such as Git, Ruby and FreeBSD first.

The learning curve for Chef is a little steep at times. Therefore, just be patient and keep working on it, and you'll eventually be done with the course. We'll certainly help you throughout the course. But, be prepared to do a lot of self-study as well. Bear in mind that the best way to learn Chef is to just use it.

Prerequisites

- Knowledge of how to use version control system, preferably Git.
- Basic Knowledge of FreeBSD OS Administration.
- Basic knowledge of any scripting language, preferably Ruby.
- Working knowledge of any text/code editor.

Resources

- FreeBSD
- FreeBSD Documentation
- Download FreeBSD
- Chef Website
- Chef Documentation
- Chef IRC Channel
- Chef Mailing List
- Git website
- Ruby Website

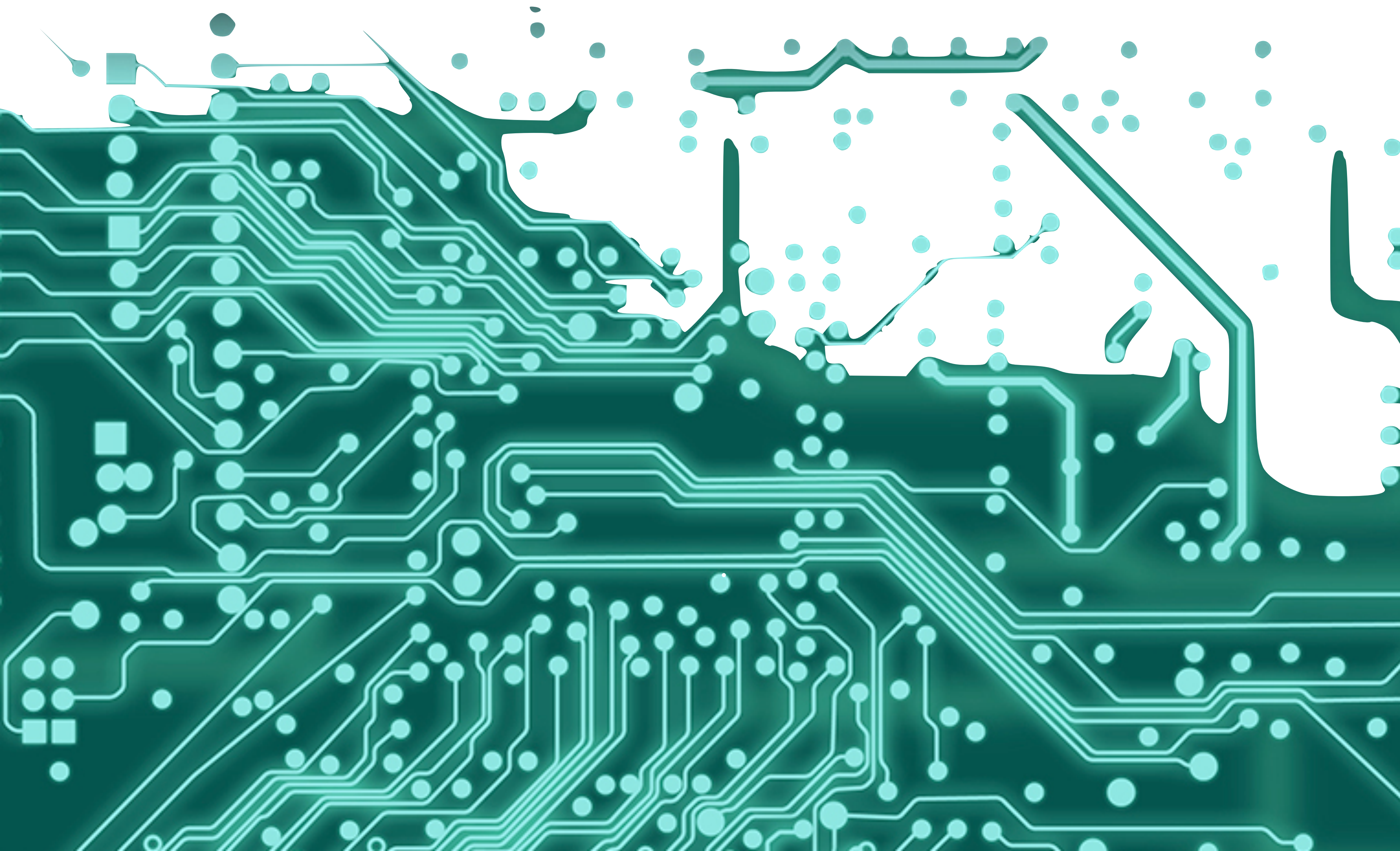
Hardware/Software Requirement

- 2 FreeBSD 11 systems/machines (Virtual or Physical), with Internet access and bash as the default shell with root login over ssh or a normal user with sudo access.
- A free account on Hosted Chef.

Course is self-paced. After completing it, you will receive a certificate with 12 CPE credits!

Join here https://bsdmag.org/product/w05-devops-chef-freebsd/

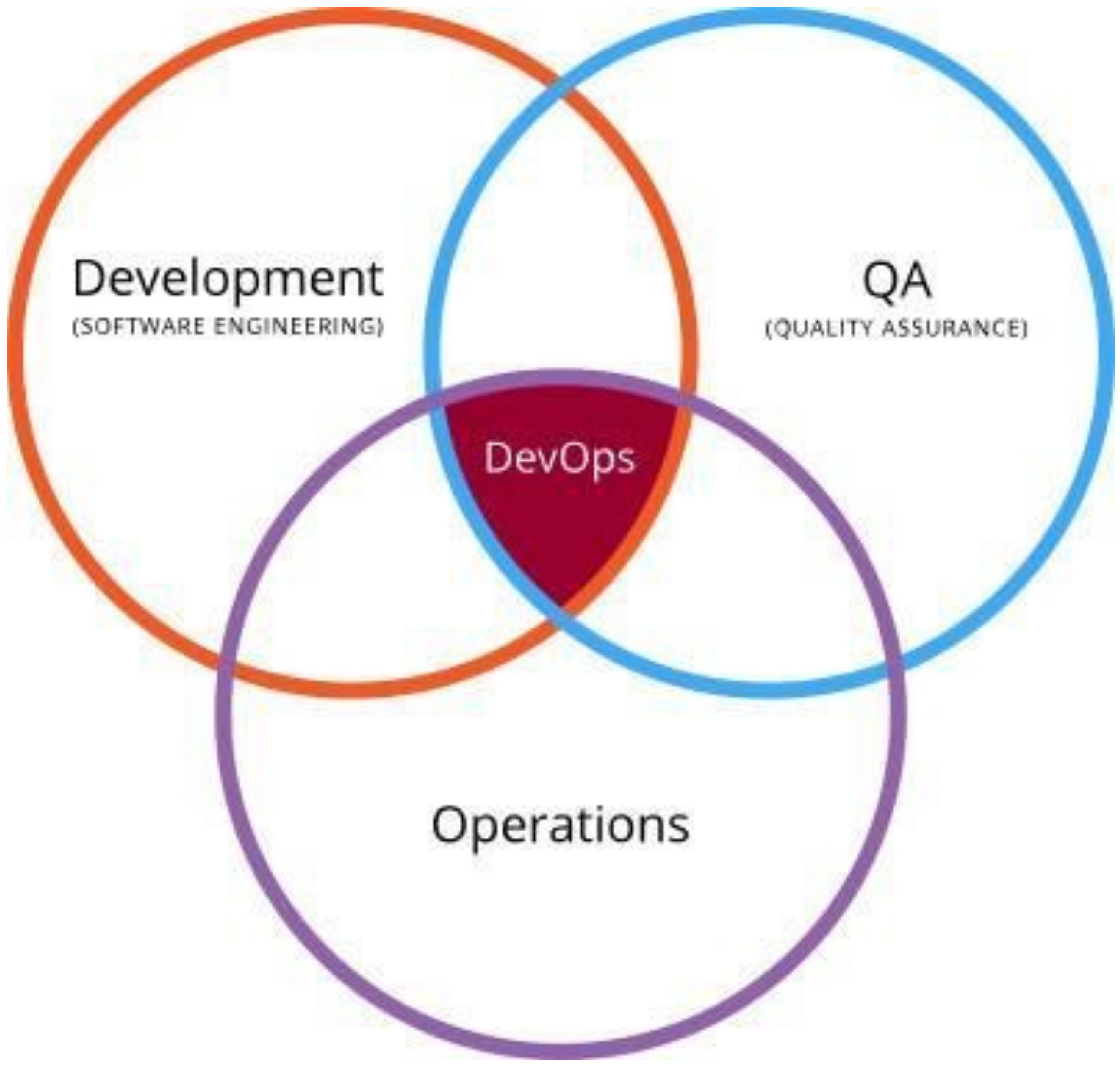
CONTENTS



Introduction to Chef

What is DevOps?

According to Wikipedia, “DevOps (a clipped compound of development and operations) is a culture, movement or practice that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructural changes.”



What is automation?

Its definition depends on the context and situation. In general, automation is a non-manual way of performing certain set of tasks. In automation of IT infrastructure, this set of tasks could be: creating new servers and infrastructure, managing existing servers, deploying applications, servers and apps, gathering metrics, graphs, scaling up or down the infrastructure and many more.

Why is it needed?

Virtualization and Cloud have forced the need for automation. In the earlier days (The Iron Age), IT infrastructure growth was dependent on the physical hardware purchasing cycle. It used to take weeks from ordering an actual server to it being delivered, and then deploying the OS and having the server ready for use.

What are the different tools/platforms available?

The most popular tools/platform available, as of now, for IT infrastructure automation are:

- Chef.
- Puppet.
- Ansible.

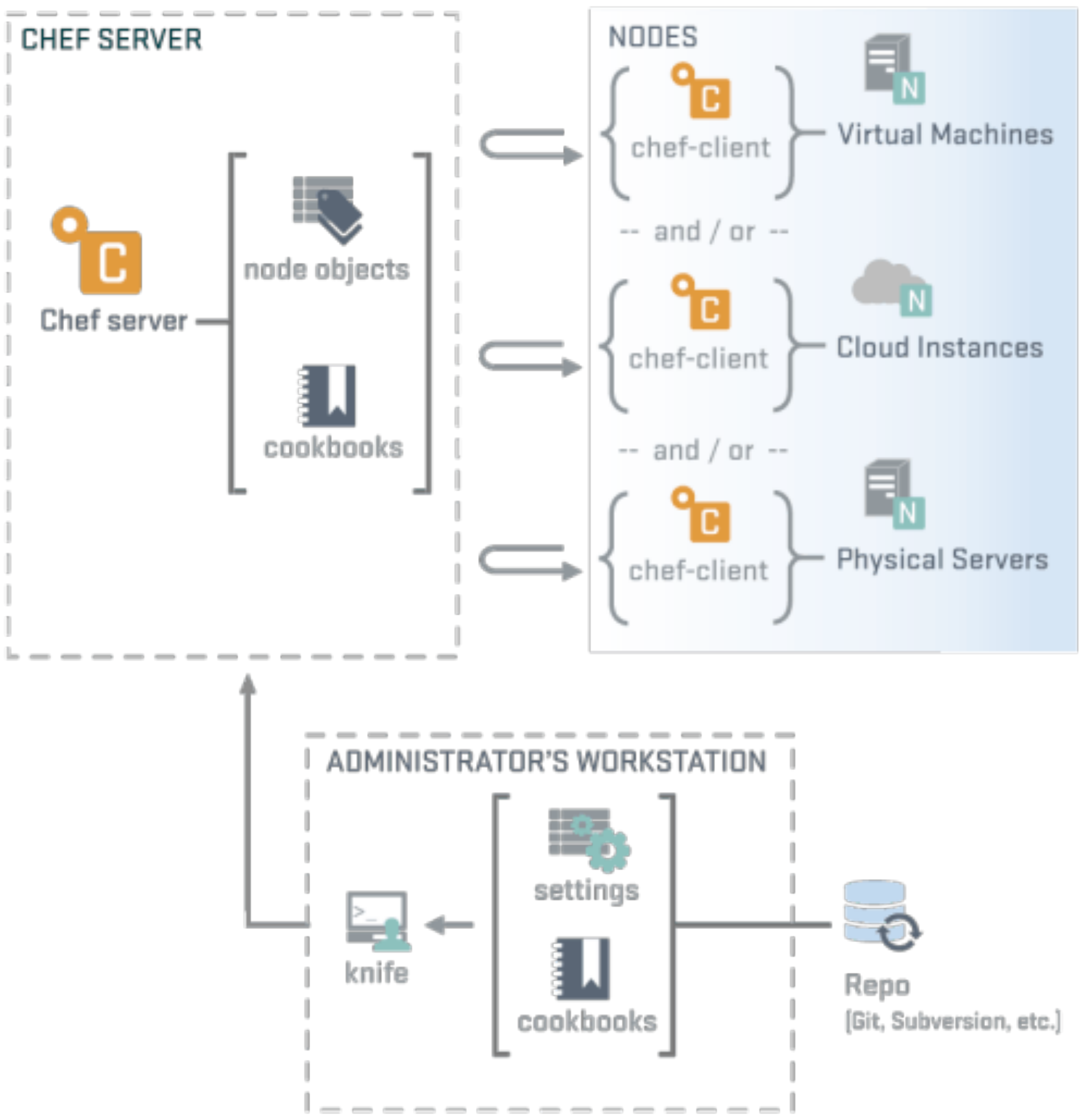
What is Chef?

Chef is a powerful automation platform that transforms infrastructure into code. It's open source. Its DSL is written in Ruby. And it's supported by almost all major Linux, Unix, Windows, cloud and container platforms. It's released under Apache2 license. The project was started in 2008, and has been growing ever since. It has a large community and is backed by a commercial for profit company/entity called "Chef".

Chef is being used by startups and Fortune 100 companies, like Facebook, GE Capital and more.

Overview of Chef.

Chef Architecture.



Chef Components

Chef Workstation

A workstation is a computer configured to run various chef command line tools, that interact with chef server and nodes. It's also used to create and maintain cookbooks.

Chef Server

A chef server acts as a single point of contact for workstation as well as the nodes/client. All the code (cookbooks/recipes) are uploaded to chef server.

Chef node

A node is any machine – physical, virtual, cloud, etc. – that is managed by chef.

Chef client

Chef client is a software or agent that runs on chef nodes. It contacts chef server periodically and downloads the code (recipes, etc) and executes them.

Chef Workstation Components

Chef repo

It's a repository structure, used to create and maintain chef cookbooks. Cookbooks contains recipes, attributes, templates, etc. This repo could be managed by version control system such as Git.

Knife

It's the command line tool to interact with chef server as well as nodes.

Chef DK

It's a package that contains everything to get started with chef. Unfortunately, it's not available for BSD as of now.

Chefspec

It's the unit test framework for chef.

Cookbook

A cookbook is a fundamental unit of configuration or policy. Following are the cookbook components:

- Attributes

Attributes can be defined in files and can be overridden. They help in making configuration dynamic.

- Recipes

It’s the most fundamental configuration element. It’s authored in ruby. It must be stored within a cookbook. It’s a collection of resources, and can be included, and can have dependencies.

- Files

Files inside the files directory are copied to the target node, just like scp.

- Libraries

It’s arbitrary ruby code to be included in a cookbook.

- Templates

A cookbook template is an embedded ruby file that’s used to dynamically generate static text file.

- Metadata

It’s a file that stores metadata about a cookbook.

Typical Chef cookbook Workflow.

- Create or Edit a cookbook.
- Upload cookbook.
- Provision machine.
- Bootstrap machine.
- Run Chef-client.
- Ssh and validate.

Resources

What is a resource?

A resource is a building block of recipes. They represent a piece of a system, e.g.: A package or directory, file, network, service, etc. It describes:

- The desired state of the configured element/item.
- Steps needed to achieve that desired state.
- Additional properties (called resource properties), if necessary.
- The type of resource.

Resource syntax:

A resource is a Ruby block with four components: a type, a name, one (or more) properties (with values), and one (or more) actions. The syntax for a resource is like this:

```
type 'name' do

   property 'value'

   action :type_of_action

end
```

Every resource has its own set of actions and properties. Most properties have default values. Some properties are available to all resources, for example, those used to send notifications to other resources and guards that help ensure that some resources are idempotent.

For example, a resource that is used to install a tar package for version 1.16.1 may look something like this:

```
package 'tar' do

  version '1.16.1'

  action :install

End
```

Common Functionality

All resources (including custom resources) share a set of common actions, properties, conditional executions, notifications, and relative path options.

Actions

Actions could be `:start, :stop, :restart, :reload` or `:nothing.`

Properties

Following properties are common to every resource: `ignore_failure, retries, retry_delay`, provider and supports

Guards

The not_if and only_if conditional executions can be used to put additional guards for certain resources, so that they are only run when the condition is met.

Notifications

Notifies and subscribes notifications can be used with any resource.

Common or most used resources

RESOURCE	DESCRIPTION
directory	Use directory resource to manage directory.
cookbook_file	Use this resource to copy file from local workstation to the node.
file	Use this to create and manage files on the node.
execute	Use this to execute a single command on the node.
git	Use it to manage resources which are in source control git repositories.
freebsd_package	Use this to manage freebsd packages.

RESOURCE	DESCRIPTION
rpm_package	Use this to manage packages using the rpm package manager.
cron	Use cron resource to manage cron entries for time based scheduling.
chef_gem	Use it to install gems only to the instance of ruby that is dedicated to chef-client.
bash	Use this to execute scripts using the bash interpreter.
package	Use it to manage packages.
user	Use this resource to add, remove, modify users and their attributes, e.g.: password, home directory, default shell, etc.
group	Use this resource to add, remove, modify group and their attributes, e.g.: group members, etc.
link	Use it to create a symbolic or hard links.
log	Use this resource to create log entries.
remote_directory	Use it to copy directories from workstation to node.
remote_file	Use this to copy/download file from a remote location to the node.
service	To manage services on the system.
apt_package	To install, remove, update, upgrade packages using apt package manager.
template	Use this to generate static content dynamically from embedded ruby text files.
yum_package	Use this resource to install, remove, update, upgrade packages using yum on Redhat or Centos based systems.
windows_package	Use this resource to manage MSI packages on Windows platform.
windows_service	Use this resource to manage windows services on Microsoft Windows platform.

Setup the workstation

Steps:

- Install ruby, rubygems and Chef on FreeBSD.

```
"` pkg install ruby ruby22-gems-2.6.4  "`

"` gem install chef "`
```

- Install vim, emacs or another text editor of your choice.

```
"` pkg install vim "`
```

- Install git version system, wget and curl

```
"` pkg install git  wget curl "`
```

- Create an account on hosted chef and download the starter kit.

  - ***https://manage.chef.io/signup/***

- Configure and test your chef workstation.

  - Unzip the starter kit
  - Change to the starter kit directory

```
"` knife client list "`
```

LAB setup (LAN)

- Network: DHCP configuration
- Workstation:
  - Address: `192.168.0.36/24`
- Node
  - Address: `192.168.0.53/24 => bsd11_node1`

- Address: `192.168.0.62/24 => bsd11_node2`
- Address: `192.168.0.39/24 => bsd11_node3`
- Gateway: `192.168.0.2`
- DNS: `8.8.8.8, 4.2.2.2`
- Default shell: `/usr/local/bin/bash`
- Ssh : `root ssh login enabled`

Install the various components mentioned above and setup the local development environment.

Bootstrap a node with Chef

Check the following before bootstrapping the node:

- Ensure that the correct hostname for the node is set.
- Ensure that wget and curl are installed on the node.
- Ensure that the internet is working on the node.
- Ensure that you can login with user root via ssh to the particular node.

Bootstrap workflow (video available in the course).

When a node (target VM, cloud, physical server) is bootstrapped, the following steps are executed:

- A client and corresponding key-pairs are created for that particular node on the Chef server.
- A node is created on the Chef server.
- Workstation connects to the target node via ssh and installs Chef-client, and configures the node.
- Converge the node (Chef-client is executed and it configures the node. This phase is also called the "Execution Phase").

On the workstation, execute the following command to bootstrap the node (target VM, cloud, physical server).

Note: All the commands need to be executed from inside the chef-repo folder, otherwise it won't work.

~~~
``` knife bootstrap  <ip address of target node > -N <name of the
target node > -x <username>  ```
~~~

E.g.:

~~~
``` knife bootstrap 192.168.0.53 -N bsd11_node1 -x root  ```
~~~

~~~
```

[test@bsd11_workstation ~/workspace/chef-repo]$ knife bootstrap
192.168.0.39 -x root -N bsd11_node3

Creating new client for bsd11_node3

Creating new node for bsd11_node3

Connecting to 192.168.0.39

Password for root@bsd11_node3.automatehub.lan:

192.168.0.39 -----> Installing Chef Omnibus (-v 12)

192.168.0.39 downloading https://omnitruck-direct.chef.io/chef/install.sh

192.168.0.39   to file /tmp/install.sh.693/install.sh

192.168.0.39 trying wget...

192.168.0.39 freebsd 11 x86_64

192.168.0.39 freebsd 11 x86_64

192.168.0.39 Getting information for chef stable 12 for freebsd...

192.168.0.39 downloading
https://omnitruck-direct.chef.io/stable/chef/metadata?v=12&p=freebsd&pv=11
&m=x86_64

192.168.0.39   to file /tmp/install.sh.697/metadata.txt

192.168.0.39 trying wget...

192.168.0.39 sha1       50ca18cc4a78c09c2cf68862876cae638e1e2689

192.168.0.39 sha256
    fe4b8ba204a57d0e1c778e358fd6e5c92891e1e427463e831e6472f1bccc864a

192.168.0.39 url
https://packages.chef.io/files/stable/chef/12.16.42/freebsd/10/chef-12.16.42_1.a
md64.sh

192.168.0.39 version    12.16.42

192.168.0.39 downloaded metadata file looks valid...

192.168.0.39 downloading
~~~

```
https://packages.chef.io/files/stable/chef/12.16.42/freebsd/10/chef-12.16.42_1.amd64.sh
192.168.0.39   to file /tmp/install.sh.697/chef-12.16.42_1.amd64.s
192.168.0.39 trying wget…
192.168.0.39
```

WARNING: could not find a valid checksum program, pre-install shasum or sha256sum in your O/S image to get validation…

```
192.168.0.39 Installing chef 12
192.168.0.39 installing with sh…
192.168.0.39 Verifying archive integrity… All good.
192.168.0.39
```

Uncompressing The full stack of chef...

```
192.168.0.39 Thank you for installing Chef!
192.168.0.39 Starting the first Chef Client run…
192.168.0.39 Starting Chef Client, version 12.16.42
192.168.0.39 resolving cookbooks for run list: []
192.168.0.39 Synchronizing Cookbooks:
192.168.0.39 Installing Cookbook Gems:
192.168.0.39 Compiling Cookbooks…
192.168.0.39 [2016-11-20T14:36:06+05:30] WARN: Node bsd11_node3 has
192.168.0.39 Compiling Cookbooks…
192.168.0.39 [2016-11-20T14:36:06+05:30] WARN: Node bsd11_node3 has
an empty run list.
192.168.0.39 Converging 0 resources
```

~~~
192.168.0.39

192.168.0.39 Running handlers:

192.168.0.39 Running handlers complete

192.168.0.39 Chef Client finished, 0/0 resources updated in 07
seconds

``` an empty run list.

192.168.0.39 Converging 0 resources

192.168.0.39

192.168.0.39 Running handlers:

192.168.0.39 Running handlers complete

192.168.0.39 Chef Client finished, 0/0 resources updated in 07
seconds

```
~~~

- Confirm that node is now showing up on the Chef server list either from the command line using the command below or from the web UI on the hosted Chef under “nodes” section/tab.

```
[test@bsd11_workstation ~/workspace/chef-repo]$ knife node list


bsd11_node1

bsd11_node2

bsd11_node3
```

Get started with writing your first cookbook

Go to chef-repo directory and initialize git repo with the following command:

~~~
``` git init ```
~~~

Add all the files in the git repo and commit to it.

```
```

git add .

Git commit -m ‘’initial commit’

```
```

Working with knife

Knife is a command-line tool that provides an interface between a local chef-repo and the Chef server. Knife helps users to manage:

- Nodes;
- Cookbooks and recipes;
- Roles;
- Stores of JSON data (data bags), including encrypted data;
- Environments;
- Cloud resources, including provisioning;
- The installation of the Chef-client on management workstations;
- Searching of indexed data on the Chef server.

For more information about knife commands and subcommands, click here >> ***https://docs.chef.io/knife.html***

Change to cookbooks directory inside the chef-repo and create a new cookbook using the following command:

~~~
[test@bsd11_workstation ~/workspace/chef-repo]$ pwd

/home/test/workspace/chef-repo

``` knife cookbook create <name of the cookbook> ```
~~~

Eg.:

~~~
``` knife cookbook create testbook```
~~~

This command will create a directory called testbook and will also create certain subdirectories and files relevant for the cookbook.

```
[test@bsd11_workstation ~/workspace/chef-repo/cookbooks]$ tree
testbook/

testbook/

|-- CHANGELOG.md

|-- README.md

|-- attributes

|-- definitions

|-- files

|   `-- default

|       `-- welcome.txt

|-- libraries

|-- metadata.rb

|-- providers

|-- recipes
```

```
|   `-- default.rb
|-- resources
`-- templates
    `-- default
```

10 directories, 5 files

Update the recipe

Next, we'll add certain code to the recipe, in this case, that code will create a directory on the target node.

```
[test@bsd11_workstation ~/workspace/chef-repo/cookbooks]$ cat
testbook/recipes/default.rb
directory "/tmp/my_tmp_dir" do
  owner "root"
  group "wheel"
  mode 0755
  action :create
end
cookbook_file "/tmp/my_tmp_dir/welcome.txt" do
  source "welcome.txt"
end
```

In the second part of the code, cookbook_file, we've created a file welcome.txt with the following content:

```
[test@bsd11_workstation ~/workspace/chef-repo/cookbooks]$ cat
testbook/files/default/welcome.txt
Welcome to Devops with Chef on FreeBSD
```

The resource cookbook_file works like scp, it'll simple copy this welcome.txt from the workstation to the target node. If the same file exists on the target node and the content is the same, it won't do anything, else, it'll create or overwrite that file.

Upload the cookbook to the Chef server.

```
[test@bsd11_workstation ~/workspace/chef-repo/cookbooks]$ knife
cookbook upload testbook

Uploading testbook       [0.1.0]

Uploaded 1 cookbook.
```

Check/list the cookbooks on the Chef server.

```
[test@bsd11_workstation ~/workspace/chef-repo/cookbooks]$ knife
cookbook list

testbook   0.1.0

vim        0.1.0
```

List all the nodes currently managed by Chef server.

```
[test@bsd11_workstation ~/workspace/chef-repo/cookbooks]$ knife node
list

bsd11_node1

bsd11_node2

bsd11_node3
```

Check the node details, especially the run list. In this case, the run list is empty.

```
[test@bsd11_workstation ~/workspace/chef-repo/cookbooks]$ knife node
show bsd11_node3

Node Name:   bsd11_node3

Environment: _default
```

```
FQDN:          bsd11_node3.automatehub.lan

IP:            192.168.0.39

Run List:

Roles:

Recipes:

Platform:      freebsd 11.0-RELEASE-p1
```

Tags:

Add the recipe "testbook" to the run list of node "`bsd11_node3`".

```
[test@bsd11_workstation ~/workspace/chef-repo/cookbooks]$ knife node
run_list add bsd11_node3 'recipe[testbook]'


bsd11_node3:

  run_list: recipe[testbook]
```

Confirm that the recipe has been added to the `node's run_list`.

```
[test@bsd11_workstation ~/workspace/chef-repo/cookbooks]$ knife node
show bsd11_node3



Node Name:     bsd11_node3

Environment:   _default

FQDN:          bsd11_node3.automatehub.lan

IP:            192.168.0.39

Run List:      recipe[testbook]

Roles:

Recipes:
```

```
Platform:    freebsd 11.0-RELEASE-p1
```

Tags:

Login to the target node and run Chef-client.

```
[root@bsd11_node3 ~]# chef-client
```

Starting Chef Client, version 12.16.42

resolving cookbooks for run list: [“testbook”]

Synchronizing Cookbooks:

– testbook (0.1.0)

Installing Cookbook Gems:

Compiling Cookbooks...

Converging 2 resources

Recipe: `testbook::default`

```
* directory[/tmp/my_tmp_dir] action create (up to date)

  * cookbook_file[/tmp/my_tmp_dir/welcome.txt] action create

    - create new file /tmp/my_tmp_dir/welcome.txt

    - update content in file /tmp/my_tmp_dir/welcome.txt from none to
2afb65

    --- /tmp/my_tmp_dir/welcome.txt 2016-11-20 15:01:53.881658000
+0530

    +++ /tmp/my_tmp_dir/.chef-welcome20161120-1136-1ynya5d.txt
2016-11-20 15:01:53.880975000 +0530

    @@ -1 +1,2 @@

    +Welcome to Devops with Chef on FreeBSD

    +Welcome to Devops with Chef on FreeBSD
```

Running handlers:

Running handlers complete

Chef Client finished, 1/2 resources updated in 09 seconds

Run the Chef client again, to see idempotence at work:

```
[root@bsd11_node3 /tmp/my_tmp_dir]# chef-client

Starting Chef Client, version 12.16.42

resolving cookbooks for run list: ["testbook"]

Synchronizing Cookbooks:

  - testbook (0.1.0)

Installing Cookbook Gems:

Compiling Cookbooks...

Converging 2 resources

Recipe: testbook::default

  * directory[/tmp/my_tmp_dir] action create (up to date)

  * cookbook_file[/tmp/my_tmp_dir/welcome.txt] action create (up to
date)
```

Running handlers:

Running handlers complete.

Chef Client finished, 0/2 resources updated in 07 seconds

Manually modify the file welcome.txt on the target node, and run the Chef-client again:

```
[root@bsd11_node3 /tmp/my_tmp_dir]# cat welcome.txt

Welcome to Devops with Chef on FreeBSD 2016

[root@bsd11_node3 /tmp/my_tmp_dir]# chef-client

Starting Chef Client, version 12.16.42
```

```
[root@bsd11_node3 /tmp/my_tmp_dir]# cat welcome.txt

Welcome to Devops with Chef on FreeBSD 2016

[root@bsd11_node3 /tmp/my_tmp_dir]# chef-client

Starting Chef Client, version 12.16.42

resolving cookbooks for run list: ["testbook"]

Synchronizing Cookbooks:

  - testbook (0.1.0)

Installing Cookbook Gems:

Compiling Cookbooks...

Converging 2 resources

Recipe: testbook::default

* directory[/tmp/my_tmp_dir] action create (up to date)

 * cookbook_file[/tmp/my_tmp_dir/welcome.txt] action create

    - update content in file /tmp/my_tmp_dir/welcome.txt from c19ce4 
to 2afb65

    --- /tmp/my_tmp_dir/welcome.txt 2016-11-20 15:03:00.536831000 
+0530

    +++ /tmp/my_tmp_dir/.chef-welcome20161120-1284-injfi3.txt 
2016-11-20 15:03:16.328688000 +0530

    @@ -1,2 +1,2 @@

    -Welcome to Devops with Chef on FreeBSD 2016

    +Welcome to Devops with Chef on FreeBSD
```

Running handlers:

```
Running handlers complete

Chef Client finished, 1/2 resources updated in 07 seconds
```

Deep dive into Chef cookbook development

A recipe is the most fundamental configuration element within the organization. A recipe:

- Is authored using Ruby;
- Is mostly a collection of resources, defined using patterns (resource names, attribute-value pairs, and actions); helper code is added around this using Ruby, when needed;
- Must define everything that is required to configure part of a system;
- Must be stored in a cookbook;
- May be included in a recipe;
- May use the results of a search query and read the contents of a data bag (including an encrypted data bag);
- May have a dependency on one (or more) recipes;
- May tag a node to facilitate the creation of arbitrary groupings;
- Must be added to a run-list before it can be used by the Chef-client;
- Is always executed in the same order as listed in a run-list.

We can write the entire code in one single `recipes/default.rb` file, but as the code grows, it becomes difficult to manage and debug. So, as a best practice, we should split the code into multiple files, and either include them in the default.rb or add them individually to the run_list as needed.

To include a recipe in the same cookbook, use the following syntax in the default.rb file.

~~~
```

Include_recipe <cookbook_name::recipe_name_without_rb_extension>
~~~

Create an apache cookbook (video available in the course content).

Attributes

An attribute is a specific detail about a node. Attributes are used by the Chef-client to understand:

- The current state of the node.
- State of the node at the end of the previous Chef-client run.
- State of the node should be at the end of current Chef-client run.

Attributes are defined by:

- The state of the node itself;
- Cookbooks (in attribute files and/or recipes);
- Roles;
- Environments.

During every chef-client run, chef-client builds the attribute list using:

- Data about the node, collected by Ohai.
- The node object that was saved to the chef server at the end of the previous chef-client run.
- The rebuilt node attribute from the current chef-client run.

After the node attribute is rebuilt, all of the attributes are compared, and then the node is updated based on the attribute precedence. At the end of every chef-client run, the node object that defines the current state of the node is uploaded to the chef server so that it can be indexed for search.

Attribute Types

There are 6 types of attributes.

ATTRIBUTE TYPE	DESCRIPTION
default	A default attribute is automatically reset at the start of every chef-client run and has the lowest attribute precedence. Use default attributes as often as possible in cookbooks.

ATTRIBUTE TYPE	DESCRIPTION
automatic	An automatic attribute contains data that is identified by Ohai at the beginning of every chef-client run. An automatic attribute cannot be modified and always has the highest attribute precedence.
force_defaul	Use force_default attribute to ensure that an attribute defined in a cookbook takes precedence over a default attribute set by a role or an environment.
normal	A normal attribute is a setting that persists in the node object. A normal attribute has a higher attribute precedence than a default attribute.
override	An override attribute is automatically reset at the start of every chef-client run and has a higher attribute precedence than default, force_default and normal attributes.
force_override	Use the force_override attribute to ensure that an attribute defined in a cookbook takes precedence over an override attribute set by a role or an environment.

Attribute Precedence

Attributes are always applied by the chef-client in the following order:

- A default attribute located in a cookbook attribute file;
- A default attribute located in a recipe;
- A default attribute located in an environment;
- A default attribute located in a role;
- A force_default attribute located in a cookbook attribute file;
- A force_default attribute located in a recipe;
- A normal attribute located in a cookbook attribute file;
- A normal attribute located in a recipe;

- An override attribute located in a cookbook attribute file;
- An override attribute located in a recipe;
- An override attribute located in a role;
- An override attribute located in an environment;
- A force_override attribute located in a cookbook attribute file;
- A force_override attribute located in a recipe;
- An automatic attribute identified by Ohai at the start of the chef-client run where the last attribute in the list is the one that is applied to the node.

Attribute Sources

Attributes are provided to the chef-client from the following locations:

- Nodes (collected by Ohai at the start of each chef-client run);
- Attribute files (in cookbooks);
- Recipes (in cookbooks);
- Environments;
- Roles.

Attribute Files

An attribute file is located in the attributes/ sub-directory for a cookbook. For example, the apache2 cookbook contains an attribute file called default.rb which contains the following attributes:

~~~
```
default['apache']['dir']  = '/etc/apache2'
default['apache']['listen_ports'] = [ '80', '443' ]
```
~~~

Templates

Resource: ***https://docs.chef.io/templates.html***

A cookbook template is an Embedded Ruby (ERB) template that is used to dynamically generate static text files. Templates may contain Ruby expressions and statements, and are a great way to manage configuration files. Use the template resource to add cookbook templates to recipes, and place the corresponding ERB template file in the cookbook's /templates directory.

Requirements

To use a template, two things must happen:

- A template resource must be added to a recipe.
- An Embedded Ruby (ERB) template must be added to a cookbook.

For example, the following template file and template resource settings can be used to manage a configuration file `named /etc/sudoers`.

Within a cookbook that uses sudo, the following resource could be added to `/recipes/default.rb`:

```
template '/etc/sudoers' do
 source 'sudoers.erb'
 mode '0440'
   owner 'root'
   group 'root'
   variables({
     sudoers_groups: node['authorization']['sudo']['groups'],
     sudoers_users: node['authorization']['sudo']['users']
   })
end
```

And then create a template called sudoers.erb and save it to `templates/default/sudoers.erb`:

```
#
# /etc/sudoers
#
# Generated by Chef for <%= node['fqdn'] %>
#
Defaults        !lecture,tty_tickets,!fqdn

# User privilege specification
root          ALL=(ALL) ALL

<% @sudoers_users.each do |user| -%>
<%= user %>    ALL=(ALL) <%= "NOPASSWD:" if @passwordless %>ALL
<% end -%>

# Members of the sysadmin group may gain root privileges
%sysadmin     ALL=(ALL) <%= "NOPASSWD:" if @passwordless %>ALL

<% @sudoers_groups.each do |group| -%>
# Members of the group '<%= group %>' may gain root privileges
%<%= group %> ALL=(ALL) <%= "NOPASSWD:" if @passwordless %>ALL
<% end -%>
```

And then set the default attributes in `attributes/default.rb`:

```
default['authorization']['sudo']['groups'] = [ 'sysadmin', 'wheel',
'admin' ]

default['authorization']['sudo']['users']  = [ 'jerry', 'greg']
```

Supermarket

Chef maintains a large collection of cookbooks. In addition, there are thousands of cookbooks created and maintained by the community.

COMPONENTS	DESCRIPTION
Cookbooks maintained by Chef	Chef maintains a collection of cookbooks that are widely used by the community.
Cookbooks maintained by the community	The community has authored and maintains thousands of cookbooks ranging from niche cookbooks to the most popular cookbooks.

The community cookbooks are available for searching via web portal ***https://supermarket.chef.io/***

Dependency Management

Resource: ***https://docs.chef.io/berkshelf.html***

Berkshelf is a dependency manager for Chef cookbooks. With it, you can easily depend on community cookbooks and have them safely included in your workflow. You can also ensure that your CI systems consistently select the same cookbook versions, and can upload and bundle cookbook dependencies without needing a locally maintained copy.

```
```

Gem install berkshelf

```
```

```
 Add the dependency in metadata.rb and Berkfile

Download the dependency

```
Berks install

```

Upload the dependency

```
```
Berks upload

```

Berksfile Syntax
```

Install BerkshelfA Berksfile is a Ruby file in which sources, dependencies, and options may be specified. Berksfiles are modeled closely on Bundler's Gemfile. The syntax is as follows:

source "*https://supermarket.chef.io*"
metadata
cookbook "`NAME" [, "VERSION_CONSTRAINT"] [, SOURCE_OPTIONS]`
```

# Advanced Chef concepts

## Roles

A role is a way to define certain patterns and processes that exist across nodes in an organization as belonging to a single job function. Each role consists of zero (or more) attributes and a run-list. Each node can have zero (or more) roles assigned to it. When a role is run against a node, the configuration details of that node are compared against the attributes of the role, and then the contents of that role's run-list are applied to the node's configuration details.

A Ruby DSL file for each role must exist in the roles/ subdirectory of the chef-repo. (If the repository does not have this subdirectory, then create it using knife.) Each Ruby file should have the .rb suffix.

The complete roles Ruby DSL has the following syntax:

```
name "role_name"

description "role_description"

run_list "recipe[name]", "recipe[name::attribute]",
"recipe[name::attribute]"

env_run_lists "name" => ["recipe[name]"], "environment_name" =>
["recipe[name::attribute]"]

default_attributes "node" => { "attribute" => [ "value", "value",
"etc." ] }

override_attributes "node" => { "attribute" => [ "value", "value",
"etc." ] }
```

Eg:

```
$ cat roles/webserver.rb

name "webserver"

description "The base role for systems that serve HTTP traffic"

run_list "recipe[apache2]", "recipe[apache2::mod_ssl]",
"role[monitor]"
```

```
env_run_lists "prod" => ["recipe[apache2]"], "staging" =>
["recipe[apache2::staging]"], "_default" => []

default_attributes "apache2" => { "listen_ports" => [ "80", "443" ] }

override_attributes "apache2" => { "max_children" => "50"
}["recipe[name::attribute]"]

default_attributes "node" => { "attribute" => [ "value", "value",
"etc." ] }

override_attributes "node" => { "attribute" => [ "value", "value",
"etc." ] }
```

Manage Roles

Upload the role to chef server:

```
$ knife role from file <path to role file.rb>
```

Eg:

Knife role from file webserver.rb

List all the roles on the chef server.

```
$ knife role list
```

Show the details of a particular role:

```
$ knife role show <role name>
```

Eg:

```
$ knife role show webserver
```

Delete the role from chef server.

Eg:

```
$ knife role delete webserver
```

Environment

Resource: ***https://docs.chef.io/environments.html***

An environment is a way to map an organization's real-life workflow to what can be configured and managed when using Chef server. Every organization begins with a single environment called the _default environment, which cannot be modified (or deleted). Additional environments can be created to reflect each organization's patterns and workflow. For example, creating production, staging, testing, and development environments. Generally, an environment is also associated with one (or more) cookbook versions.

Define an Environment

A Ruby file for each non-default environment must exist in the environments/ subdirectory of the chef-repo. (If the chef-repo does not have this subdirectory, then it should be created.) The complete environment has the following syntax:

```
name 'environment_name'

description 'environment_description'

cookbook OR cookbook_versions  'cookbook' OR 'cookbook' =>
'cookbook_version'

default_attributes 'node' => { 'attribute' => [ 'value', 'value',
'etc.' ] }

override_attributes 'node' => { 'attribute' => [ 'value', 'value',
'etc.' ] }
```

where both default and override attributes are optional and either a cookbook or cookbook versions (one or more) are specified. For example, an environment named dev that uses the couchdb cookbook (version 11.0.0 or higher) that listens on ports 80 and 443:

```
name 'dev'

description 'The development environment'

cookbook_versions  'couchdb' => '= 11.0.0'

default_attributes 'apache2' => { 'listen_ports' => [ '80', '443' ] }
```

Manage Environments

Upload the environment to chef server:

```
$ knife environment from file <path to env.rb file>
```

Eg:

```
$ knife environment from file dev.rb
```

List the environments on the chef server.

```
$ knife environment list
```

Show the details of a particular environment:

```
$ knife environment show <name of the env>
```

Eg:

```
$ knife environment show dev
```

Delete the environment from Chef server:

```
$ knife environment delete dev
```

Search

Resource: ***https://docs.chef.io/chef_search.html***

Search indexes allow queries to be made for any type of data that is indexed by the Chef server, including data bags (and data bag items), environments, nodes, and roles. A defined query syntax is used to support search patterns like exact, wildcard, range, and fuzzy. A search is a full-text query that can be done from several locations, including from within a recipe, by using the search subcommand in knife, the search method in the Recipe DSL, the search box in the Chef management console, and by using the /search or /search/INDEX endpoints in the Chef server API. The search engine is based on Apache Solr and is run from the Chef server.

Search Indexes

SEARCH INDEX NAME	DESCRIPTION
client	API client
DATA_BAG_NAME	A data bag is a global variable that is stored as JSON data and is accessible from a Chef server. The name of the search index is the name of the data bag. For example, if the name of the data bag was “admins” then a corresponding search query might look something like search(:admins, “*:*”).
Environment	An environment is a way to map an organization’s real-life workflow to what can be configured and managed when using Chef server.
node	A node is any server or virtual server that is configured to be maintained by a chef-client.
Role	A role is a way to define certain patterns and processes that exist across nodes in an organization as belonging to a single job function.

Usage

Search by node

To search for all nodes running Ubuntu, enter:

```
$ knife search node 'platform:ubuntu'
```

Search by node and environment

To search for all nodes running CentOS in the production environment, enter:

```
$ knife search node 'chef_environment:production AND platform:centos'
```

Query Syntax

A search query is comprised of two parts: the key and the search pattern. A search query has the following syntax:

```
key:search_pattern
```

where key is a field name that is found in the JSON description of an indexable object on the Chef server (a role, node, client, environment, or data bag) and search_pattern defines what will be searched for, using one of the following search patterns: exact, wildcard, range, or fuzzy matching. Both key and search_pattern are case-sensitive; key has limited support for multiple character wildcard matching using an asterisk ("*") (and as long as it is not the first character).

Using search with recipes

The following examples show how a recipe can use a series of search queries to search within a data bag named "admins".

For example, to find every administrator:

```
search(:admins, "*:*")
```

Or to search for an administrator named "charlie":

```
search(:admins, "id:charlie")
```

Or to search for an administrator with a group identifier of "ops":

```
search(:admins, "gid:ops")
```

Or to search for an administrator whose name begins with the letter "c":

```
search(:admins, "id:c*")
```

Databags

Resource: ***https://docs.chef.io/data_bags.html***

A data bag is a global variable that is stored as JSON data and is accessible from a Chef server. A data bag is indexed for searching and can be loaded by a recipe or accessed during a search.

Create a Data Bag

A data bag can be created in two ways: using knife or manually. In general, using knife to create data bags is recommended, but as long as the data bag folders and data bag item JSON files are created correctly, either method is safe and effective.

Using knife

knife can be used to create data bags and data bag items when the knife data bag subcommand is run with the create argument. For example:

```
$ knife data bag create DATA_BAG_NAME (DATA_BAG_ITEM)
```

knife can be used to update data bag items using the from file argument:

As long as a file is in the correct directory structure, knife will be able to find the data bag and data bag item with only the name of the data bag and data bag item. For example:

will load the following file:

```
data_bags/BAG_NAME/ITEM_NAME.json
```

Store Data in a Data Bag

When the chef-repo is cloned from GitHub, the following occurs:

- A directory named data_bags is created.
- For each data bag, a sub-directory is created that has the same name as the data bag.
- For each data bag item, a JSON file is created and placed in the appropriate sub-directory.

The data_bags directory can be placed under version source control.

DataBag Item

A data bag is a container of related data bag items, where each individual data bag item is a JSON file. knife can load a data bag item by specifying the name of the data bag to which the item belongs and then the filename of the data bag item.

The only structural requirement of a data bag item is that it must have an id:

```
{
  "id": "ITEM_NAME",
  "key": "value"
}
```

where key and value are the key:value pair for each additional attribute within the data bag item.

Manage data bag items

Create a data bag item from a file

Syntax:

knife data bag from file `BAG FILE|FOLDER [FILE|FOLDER..]`

Eg:

```
$ knife data bag from file users automatehub.json
```

List all the data bag:

```
$ knife data bag list
```

Show data bag item

Syntax:

```
$ knife data bag show BAG [item]
```

Eg:

```
$ knife data bag show users automatehub
```

Delete data bag item

Syntax:

knife data bag delete `BAG [ITEM]`

Eg:

```
$ knife data bag delete users automatehub
```

Using syntax and linting tools like: Foodcritic and RuboCop

Foodcritic

Resources: ***http://www.foodcritic.io/***

Foodcritic is a linting tool that you can use to check your chef cookbooks for common problems.

- It comes with 61 built in rules to help you identify simple style inconsistencies to hard to debug issues.
- There are community rules as well that you can download and use.
- Or write your own.

Use Foodcritic to check cookbooks for common problems:

- Style;
- Correctness;
- Syntax;
- Best practices;
- Common mistakes;
- Deprecations.

Foodcritic returns a list, via standard output, that shows the results of the evaluation:

FC003: Check whether you are running with chef server before using server-specific features: `./recipes/ip-logger.rb:1`

```
FC008: Generated cookbook metadata needs updating: ./metadata.rb:2
FC008: Generated cookbook metadata needs updating: ./metadata.rb:3
Output¶
```

Foodcritic output:

States a Foodcritic rule. For example: `FC003`.

Describes the rule, based on the results of the evaluation. For example: Check whether you are running with chef server before using server-specific features:

Specifies the file path. For example: `./recipes/ip-logger.rb`

Specifes the line number. For example: `1`

A Foodcritic evaluation has the following syntax:

```
RULENUMBER: MESSAGE: FILEPATH:LINENUMBER
```

For example:

```
FC008: Generated cookbook metadata needs updating: ./metadata.rb:3
```

Getting started is really easy.

~~~
``` gem install foodcritic ```

[root@bsd11_workstation ~]# gem install foodcritic

Fetching: gherkin-4.0.0.gem (100%)

Successfully installed gherkin-4.0.0

Fetching: backports-3.6.8.gem (100%)

Successfully installed backports-3.6.8

Fetching: cucumber-core-2.0.0.gem (100%)

Successfully installed cucumber-core-2.0.0

Fetching: mini_portile2-2.1.0.gem (100%)

Successfully installed mini_portile2-2.1.0

Fetching: nokogiri-1.7.0.1.gem (100%)

Building native extensions.  This could take a while...

Successfully installed nokogiri-1.7.0.1
~~~

```
Fetching: rake-12.0.0.gem (100%)

Successfully installed rake-12.0.0

Fetching: polyglot-0.3.5.gem (100%)

Successfully installed polyglot-0.3.5

Fetching: treetop-1.6.8.gem (100%)

Successfully installed treetop-1.6.8

Fetching: yajl-ruby-1.3.0.gem (100%)

Building native extensions.  This could take a while...

Successfully installed yajl-ruby-1.3.0

Fetching: rufus-lru-1.1.0.gem (100%)

Successfully installed rufus-lru-1.1.0

Fetching: foodcritic-8.2.0.gem (100%)

Successfully installed foodcritic-8.2.0

Parsing documentation for gherkin-4.0.0

Installing ri documentation for gherkin-4.0.0

Parsing documentation for backports-3.6.8

Installing ri documentation for backports-3.6.8

Parsing documentation for cucumber-core-2.0.0

Installing ri documentation for cucumber-core-2.0.0

Parsing documentation for

mini_portile2-2.1.0

Installing ri documentation for mini_portile2-2.1.0

Parsing documentation for nokogiri-1.7.0.1

Installing ri documentation for nokogiri-1.7.0.1

Parsing documentation for rake-12.0.0

Installing ri documentation for rake-12.0.0
```

```
Parsing documentation for polyglot-0.3.5
Installing ri documentation for polyglot-0.3.5
Parsing documentation for treetop-1.6.8
Installing ri documentation for treetop-1.6.8
Parsing documentation for yajl-ruby-1.3.0
Installing ri documentation for yajl-ruby-1.3.0
Parsing documentation for rufus-lru-1.1.0
Installing ri documentation for rufus-lru-1.1.0
Parsing documentation for foodcritic-8.2.0
Installing ri documentation for foodcritic-8.2.0
```

Install Foodcritic.Done installing documentation for gherkin, backports, cucumber-core, mini_portile2, nokogiri, rake, polyglot, treetop, yajl-ruby, rufus-lru, foodcritic after 25 seconds

11 gems installed

Run Foodcritic

~~~
``` foodcritic <path to your cookbook > ```
~~~

When Foodcritic encounters a rule/policy violation, it'll report that rule number along with brief text.

RuboCop

Resources: ***https://docs.chef.io/rubocop.html***

Most of the code that is authored when working with Chef is written as Ruby. Just about every file within a cookbook—with few exceptions!—is a Ruby file.

Use RuboCop to author better Ruby code:

- Enforce style conventions and best practices.
- Evaluate the code in a cookbook against metrics like "line length" and "function size".
- Help every member of a team to author similarly structured code.

- Establish uniformity of source code.
- Set expectations for fellow (and future) project contributors.

RuboCop is a style and linting tool that analyzes all of the Ruby code that is authored in a cookbook against a number of rules. (RuboCop rules are sometimes referred to as "cops".) Each rule examines the code for a specific perspective, after which a list of warnings, deviations from patterns, potential errors, and fatal errors is generated.

RuboCop is built for Ruby developers by Ruby developers. As such, RuboCop will enforce the conventions that are defined by that community. As users of Chef and as authors of cookbooks, even though we are using Ruby, we do not always have the same objectives and goals. That said, there is enough of an overlap that using RuboCop as part of a cookbook authoring workflow is beneficial.

Each rule in RuboCop may be enabled and disabled. Custom rules may be created to assist with enforcing standards that are unique to any cookbook authoring team.

Run RuboCop

RuboCop is run from the command line, typically against a single cookbook and all of the

Ruby files contained within it:

```
$ rubocop /path/to/cookbook
```

RuboCop may also be run from the root of an individual cookbook directory:

```
$ rubocop .
```

RuboCop returns a list, via standard output, that shows the results of the evaluation:

Symbols

The following symbols appear in the standard output and are used to indicate the result of an evaluation:

SYMBOL	DESCRIPTION
client	API client
.	The file does not have any issues.
C	The file has an issue with convention.
E	The file contains an error.
F	The file contains a fatal error.
W	The file contains a warning.

```
[root@bsd11_workstation ~]# gem install rubocop
Fetching: rainbow-2.2.1.gem (100%)
Building native extensions.  This could take a while...
Successfully installed rainbow-2.2.1
Fetching: ast-2.3.0.gem (100%)
Successfully installed ast-2.3.0
Fetching: parser-2.3.3.1.gem (100%)
Successfully installed parser-2.3.3.1
Fetching: powerpack-0.1.1.gem (100%)
Successfully installed powerpack-0.1.1
Fetching: ruby-progressbar-1.8.1.gem (100%)
Successfully installed ruby-progressbar-1.8.1
Fetching: unicode-display_width-1.1.3.gem (100%)
Successfully installed unicode-display_width-1.1.3
Fetching: rubocop-0.47.1.gem (100%)
```

```
Successfully installed rubocop-0.47.1

Parsing documentation for rainbow-2.2.1

Installing ri documentation for rainbow-2.2.1

Parsing documentation for ast-2.3.0

Installing ri documentation for ast-2.3.0

Parsing documentation for parser-2.3.3.1

Installing ri documentation for parser-2.3.3.1

Parsing documentation for powerpack-0.1.1

Installing ri documentation for powerpack-0.1.1

Parsing documentation for ruby-progressbar-1.8.1

Installing ri documentation for ruby-progressbar-1.8.1

Parsing documentation for unicode-display_width-1.1.3

Installing ri documentation for unicode-display_width-1.1.3

Parsing documentation for rubocop-0.47.1

Installing ri documentation for rubocop-0.47.1
```

Done installing documentation for rainbow, ast, parser, `powerpack`, `ruby-progressbar`, `unicode-display_width`, rubocop after 40 seconds.

7 gems installed.

Writing unit test and integration tests.

ChefSpec – Unit Testing

Resources:

https://github.com/sethvargo/chefspec/tree/master/examples

https://docs.chef.io/chefspec.html

Unit Testing is a software testing method by which individual units of the code are tested. Advantages of Unit Testing:

- Help in finding problems early in the development cycle.
- Facilitates changes.
- Simplifies integration by reducing uncertainty.

ChefSpec is a framework that tests resources and recipes as part of a simulated chef-client run. ChefSpec tests execute very quickly. When used as part of the cookbook authoring workflow, ChefSpec tests are often the first indicator of problems that may exist within a cookbook.

ChefSpec

- Is an extension of RSpec, a behavior-driven development (BDD) framework for Ruby.
- Is the fastest way to test resources and recipes.

Note: ChefSpec simulates the execution of your resources in memory, and does not involve the creation of a virtual instance. It's the fastest way to test your resources and is a great way to validate the correctness of your work, even before you set up a virtual instance to test on.

```
Install ChefSpec

```

Gem install chefspec

```
```

If you use chef cli executable (recommended) to generate cookbooks, recipes, templates, attributes, etc., then, for every recipe created, a spec file is also created automatically.

You could also create the directory structure and required files manually, if needed.

You typically define one test, or spec, file for each recipe. So `default_spec.rb` maps to the default recipe, `default.rb`. If you had a second recipe, say firewall.rb, then you would have a spec named `firewall_spec.rb`.

```
tree apache/
apache/
|-- Berksfile
|-- Berksfile.lock
|-- README.md
|-- attributes
|   `-- default.rb
|-- chefignore
|-- metadata.rb
|-- recipes
|   |-- config.rb
|   |-- default.rb
|   `-- install.rb
|-- spec
|   |-- spec_helper.rb
|   `-- unit
|       `-- recipes
|           |-- config_spec.rb
|           |-- default_spec.rb
|           `-- install_spec.rb
```

```
|-- templates
|   `-- default
|       `-- index.html.erb
`-- test
    `-- smoke
        `-- default
            |-- config.rb
            |-- default_test.rb
            `-- install.rb
```

Let's have a look at `default_spec.rb`:

```
$ pwd
/home/test/workspace/chef-repo/cookbooks
$ cat apache/spec/unit/recipes/default_spec.rb
#
# Cookbook:: apache
# Spec:: default
#
# Copyright:: 2017, The Authors, All Rights Reserved.
require 'spec_helper'
describe 'apache::default' do
  context 'When all attributes are default, on an unspecified
platform' do
    let(:chef_run) do
      runner = ChefSpec::ServerRunner.new
      runner.converge(described_recipe)
    end
    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end
  end
end
```

The describe blocks tells ChefSpec to run the apache::default recipe in memory. The let block simulates the chef-client run. It also defines the chef_run variable, which is used in each test to validate the result.

ChefSpec::ServerRunner specifies how to run chef-client in memory. It's a common option because it also simulates an in-memory Chef server, allowing you to access data bags *https://docs.chef.io/data_bags.html* and other features.

A great way to learn how to write good tests is by example. The ChefSpec project contains example tests *https://github.com/sethvargo/chefspec/tree/master/examples* for many common Chef resource types.

Run your tests, using the executable 'rspec'.

```
```

$ cd <path to cookbook>
$ rspec <absolute or relative path to the spec file >
Or
(to run all the tests in the specs directory)
$ rspec .
```
```

Run your tests, using the executable 'rspec'.

Inspec – Integration Testing/Auditing

Resources: *http://inspec.io/#*

Module 9

Integration Testing is the phase in software testing in which individual software modules/units are combined and tested as a group. This phase occurs after unit testing.

Audit means an independent examination of a software product or processes to assess compliance with specifications, standards, contractual agreements, or other criteria.

InSpec is an open-source run-time framework and rule language used to specify compliance, security, and policy requirements for testing any node in your infrastructure.

- The project name refers to “infrastructure specification”;
- InSpec includes a collection of resources to help you write auditing rules quickly and easily using the Compliance DSL;
- Use InSpec to examine any node in your infrastructure; run the tests locally or remotely;
- Any detected security, compliance, or policy issues are flagged in a log.

InSpec provides a kind of integration testing/auditing, where you verify that multiple components function correctly together. With InSpec, you write code that describes the desired state of the server. InSpec translates this code into SSH commands that it runs on the server.

It’s common to write InSpec tests after you write your configuration code. But another way is to take a test-driven approach, where you write your tests first before you write any Chef code.

The idea behind test-driven development is to use tests, also called specifications or specs, to clearly document your requirements. You run your tests on the server and watch them each fail. Then you write just enough Chef code to make at least one failing test pass. You then repeat the process until all of your tests pass.

InSpec is:

- Platform agnostic;
- Free to run anywhere;
- Test systems locally or remotely.

An important benefit of test-driven development is that it helps limit scope. You know you’re done when all tests pass. And because the tests are code, they’re versionable through source control. When a requirement changes, you capture that change by updating the tests. This gives you the complete history of your requirements and better insight into when and how your requirements changed.

Module 9

Here you'll create a basic cookbook that resembles the Apache web server configuration that you wrote previously. Let's begin by defining clear goals for the web server configuration.

Here's the criteria for the web server configuration:

- Install the Apache package.
- Serve a custom home page.

Install inspec:

```
```

Gem install inspec

```
```

Write tests:

```
[test@bsd11_workstation ~/workspace/chef-repo/cookbooks]$ tree
apache/

apache/

|-- Berksfile

|-- Berksfile.lock

|-- README.md

|-- attributes

|   `-- default.rb

|-- chefignore

|-- metadata.rb

|-- recipes

|   |-- config.rb

|   |-- default.rb

|   `-- install.rb

|-- spec
```

```
|   |-- spec_helper.rb
|   `-- unit
|       `-- recipes
|           |-- config_spec.rb
|           |-- default_spec.rb
|           `-- install_spec.rb
|-- templates
|   `-- default
|       `-- index.html.erb
`-- test
    `-- smoke
        `-- default
            |-- config.rb
            |-- default_test.rb
            `-- install.rb
```

10 directories, 17 files

The file `test/smoke/default/default_test.rb` will hold our tests. You typically have one test file for each recipe in your cookbooks. Here, `default_test.rb` contains tests for the default recipe, default.rb.

For this tutorial, we'll add all the configuration code to the install recipe.

As with many test frameworks, InSpec code resembles natural language. Here's the format of an InSpec test.

```
describe '<entity>' do
  it { <expection> }
end
```

An InSpec test has two main components: the subject to examine and the subject’s expected state. Here, <entity> is the subject you want to examine, for example, a package name, service, file, or network port. The <expectation> part specifies the desired result or expected state, for example, that a port should be open (or perhaps should not be open).

Example:

```
[test@bsd11_workstation ~/workspace/chef-repo/cookbooks]$ cat
apache/test/smoke/default/install.rb

# # encoding: utf-8

# Inspec test for recipe apache::install

# The Inspec reference, with examples and extensive documentation,
can be

# found at http://inspec.io/docs/reference/resources/

# check if the package is installed or not

describe package('apache24') do

  it { should be_installed }

end

# check port

describe port(80) do

  it { should be_listening }

  its('processes') { should include 'httpd' }

  its('protocols') { should include 'tcp' }

  its('addresses') { should include '0.0.0.0' }

end

# confirm if apache is running

describe service('apache24') do

  it { should be_running }

end
```

```
# confirm the output

describe command('curl http://localhost') do

its(:stdout) { should match /Welcome to Devops with Chef on FreeBSD/
}

end

# check user

describe user('automatehub') do

  it { should exist }

end

Run "inspec exec " cli executable to run the tests.
```

About the Instructor:

Arun Tomar is an Entrepreneur, Technology evangelist, Solution Architect, Consultant & Corporate Trainer with deep expertise in designing and providing end to end solutions for enterprise IT Infrastructure requirements. He has more than 12 years of experience. He is currently the Director and CTO at AutomateHub Service, Inc., Canada.

Course is self-paced. After completing it, you will receive a certificate with 12 CPE credits!
Join here https://bsdmag.org/product/w05-devops-chef-freebsd/